Anatomie eines Compilers am Beispiel von Pugs

Ingo Blechschmidt
<iblech@web.de>

LUGA

3. August 2005

Inhalt

Pugs

- Pugs: Prototyp des Perl 6-Compilers
- „Perl 6 ist ja schön und gut, aber das dauert doch noch Jahre, bis es fertig ist!“
- „Die Entwickeln doch schon seit Jahren dran!“
- Nur tote Produkte sind „fertig“.
- Seit dem 1. Februar gibt es Pugs. Heute kann man vernünfig in Perl 6 programmieren.

Pugs

- Ursprünglich Haskell-Projekt von Autrijus Tang „als Übung“
- Projektbeginn: 1. Februar 2005
- Nun 130 Entwickler
- Version 6.2.8: Beinahe Alles (!), mehrere Backends (direkte Ausführung, Kompilierung zu Haskell, zu Perl 5, zu JavaScript, etc.)

Entwicklung

- Test-driven development –
- Camelfolk: Schreiben von Tests in Perl 6 für noch nicht implementierte Features

```
is 23 + 42, 64, "Einfache Rechnungen funzen.";

my @array = <a b c>;
is +@array, 3,
  "Unser Array enthält drei Elemente.";
```

- Lambdafolk: Implementierung dieser Features
- Ergebnis der Zusammenarbeit:
  Über 7.700 funktionierende Tests

Entwicklung

- Test-driven development –
- Camelfolk: Schreiben von Tests in Perl 6 für noch nicht implementierte Features

```
is 23 + 42, 64, "Einfache Rechnungen funzen.";

my @array = <a b c>;
is +@array, 3,
  "Unser Array enthält drei Elemente.";
```

- Lambdafolk: Implementierung dieser Features
- Ergebnis der Zusammenarbeit:
  Über 7.700 funktionierende Tests

Pläne

Pugs 6.0	Erstes Release
Pugs 6.2	Grundlegende IO- und Kontrollflusselemente, veränderbare Variablen
Pugs 6.28	Klassen
Pugs 6.283	Rules und Grammars
Pugs 6.2831	Rollen
Pugs 6.28318	Makros
Pugs 6.283185	Portierung von Pugs von Haskell nach Perl 6
Pugs 2π	Vollendung

Beteiligungsmöglichkeiten

- Mailinglisten:
  `perl6-language@perl.org`,
  `perl6-compiler@perl.org`,
  `gmane.comp.lang.perl.perl6.language`,
  `gmane.comp.lang.perl.perl6.compiler`
- IRC: `#perl6` auf Freenode
- Auch Newbies sehr gern gesehen!
- Schreiben von Tests (Perl 6), Implementierung (Haskell), Schreiben von Dokumentation, Portierung von Perl 5|Python|Ruby|...-Modulen nach Perl 6, ...
- Weitere Informationen: `http://www.pugscode.org/`

Arbeitsschritte

1. Parsen: Umwandlung des Sourcecode in einen Parse Tree
2. Kleinere Optimierungen
3. Umwandlung des Parse Tree in einen einfacheren Tree
4. Größere Optimierungen, Argumentieren über den Code (z.B. Verbot von `3 = 4` zur Compile-Zeit)
5. Umwandlung ins Zielformat
6. Kleinere Optimierungen
7. Ausgabe

Beispiel: Perl 6 → JavaScript-Compiler

- PIL2JS: Spiel-Projekt von mir, Projektbeginn: 16.7.2005
- ≈ 4.000 Zeilen Perl 5, Perl 6 und JavaScript
- „Perl 6 überall“ (Browser, PDFs, Flash, . . .)

1. Einlesen und Parsen von Perl 6 durch Pugs
2. Ausgabe von Pugs Intermediate Language (PIL) durch Pugs
3. Einlesen des PIL-Trees durch PIL2JS
4. Kleinere Umwandlungen
5. Ausgabe als JavaScript

Beispiel: Perl 6 → JavaScript-Compiler

- PIL2JS: Spiel-Projekt von mir, Projektbeginn: 16.7.2005
- ≈ 4.000 Zeilen Perl 5, Perl 6 und JavaScript
- „Perl 6 überall“ (Browser, PDFs, Flash, . . .)

1. Einlesen und Parsen von Perl 6 durch Pugs
2. Ausgabe von Pugs Intermediate Language (PIL) durch Pugs
3. Einlesen des PIL-Trees durch PIL2JS
4. Kleinere Umwandlungen
5. Ausgabe als JavaScript

Parsen von Perl 6-Sourcecode (Perl 6 → PIL)

- Perl 6 ist eine umfangreiche Sprache.
- Wenn jedes Backend Perl 6 selbst parsen müsste, wäre das viel doppelte Arbeit.
- Stattdessen: Parsen von Perl 6 durch Pugs, Ausgabe des Codes in einer Zwischen-Sprache, Pugs Intermediate Language (PIL)
- Einlesen des PIL durch die einzelnen Backends – Kümmern ums Parsen unnötig

Beispiel

```
# Perl 6:
$foo = 19;
say 4 + $foo;
```

```
-- PIL (vereinfacht):
PAssign (PVar "$foo") (PLit 19)
PApp (PVar "&say") [
  PApp (PVar "&infix:<+>") [
    PLit 4, PVar "$foo"
  ]
]
```

Kompilieren des PIL zu JavaScript

- „Sowohl Perl 6 als auch JavaScript sind Turing-vollständig, wo also liegt das Problem? (:D)“
- JavaScript: weniger mächtig als Perl 6
- Also: Herunterkompilation vieler Features erforderlich

Problem: Signaturen von Subroutinen

- Perl 6: Reiche Möglichkeiten zur Spezifikation von Signaturen (Parameter-Listen; ähnlich wie Ruby oder Python):

```
sub foo (Grtz $grtz, Bool ?$verbose = false) {...}

# Ok:
foo $irgendein_grtz_objekt;
foo $irgendein_grtz_objekt, true;
foo $irgendein_grtz_objekt, :verbose;
foo $irgendein_grtz_objekt, :verbose(true);
foo $irgendein_grtz_objekt, verbose => true;

# Fehler:
foo "Zu", <viele>, $parameter;
foo();
```

Problem: Signaturen von Subroutinen

- JavaScript (vor Version 2): Weit weniger umfangreiche Möglichkeiten, Ignorieren von zu vielen/zu wenigen Parametern (ähnlich wie PHP):

```
function foo (grtz, verbose) {...}

// Ok:
foo(irgendein_grtz_objekt);
foo(irgendein_grtz_objekt, true);

// Ebenfalls ok (!):
foo();
foo("Zu", viele, Para.meter);
```

Problem: Lexikale Variablen

- Perl 6: Lexikale Variablen (wie bei Ruby, Python, C und vielen anderen Sprachen):

```
{ say $a }              # Fehler
{ my $a; say $a }       # Ok
{ say $a; my $a }       # Fehler
```

- JavaScript (ähnlich wie bei Bash oder PHP):

```
{ alert(a) }            // Fehler
{ var a; alert(a) }     // Ok
{ alert(a); var a }     // Kein (!) Fehler
```

- Daher, leider: Durchnummerieren aller lexikalen Variablen ($a_1, $a_2, ...) und dann Deklaration als globale JavaScript-Variablen

Problem: Objekt-Metamodell

Objekt-Metamodell

„Was ist eine Klasse?“ – „Was ist ein Objekt?“ – „Ist eine Klasse auch ein Objekt?“ – . . .

- Perl 6: Mächtiges Objekt-Metamodell, mit Features u.a. von Smalltalk und CLOS
- JavaScript: Weniger mächtiges Modell

- Viele Backends haben dieses Problem.
- Daher: Exzellente Arbeit von Stevan Little:
  Perl 6-Metamodell für Perl 5, Perl 6, JavaScript, Java, C#, . . .

Weitere Probleme

- Firefox: langsame JavaScript-Ausführung
- Wichtiger noch: Ausführung von Seiten-JavaScripts im gleichen Thread wie die UI (Hänger!)
- Aber: Exzellente JavaScript-Implementation

„Never do any live demos!“

- Hello, World!
- `mandel.p6`
- Testsuite

Fazit

- Compiler-Schreiben ist leichter als man denkt. :D
- Besonders leicht wird es, wenn einem viel Arbeit abgenommen wird. :)

Fazit

- Compiler-Schreiben ist leichter als man denkt. :D
- Besonders leicht wird es, wenn einem viel Arbeit abgenommen wird. :)

Join the fun!

`http://www.pugscode.org/`